



20450 Century Boulevard
Germantown, MD 20874

MCASP LLD

Software Design Specification (SDS)

Revision A

Document License

This work is licensed under the Creative Commons Attribution-NoDerivs 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contributors to this document

Copyright (C) 2012 Texas Instruments Incorporated - <http://www.ti.com/>

Revision Record	
Document Title: Software Design Specification	
Revision	Description of Change
A	1. Initial Release – Code drop 1.1.0

Note: Be sure the Revision of this document matches the Approval record Revision letter. The revision letter increments only upon approval via the Quality Record System.

TABLE OF CONTENTS

1 SCOPE2

2 REFERENCES2

3 DEFINITIONS2

4 OVERVIEW3

4.1 HARDWARE OVERVIEW3

4.2 SOFTWARE OVERVIEW5

4.3 KEY FEATURES6

5 DESIGN6

5.1 MCASP DRIVER INITIALIZATION7

5.2 MCASP PERIPHERAL CONFIGURATION7

5.3 MCASP DRIVER EXTERNAL INTERFACE (PUBLIC APIS).....9

5.3.1 Driver Instance Binding.....10

5.3.2 Channel Creation.....12

5.3.3 I/O Frame Processing.....14

 5.3.3.1 Asynchronous I/O Mechanism.....14

5.3.4 Control Commands.....14

 5.3.4.1 Mcasp_IOCTL_DEVICE_RESET.....14

 5.3.4.2 Mcasp_IOCTL_CNTRL_AMUTE15

 5.3.4.3 Mcasp_IOCTL_START_PORT.....15

 5.3.4.4 Mcasp_IOCTL_STOP_PORT15

 5.3.4.5 Mcasp_IOCTL_QUERY_MUTE16

 5.3.4.6 Mcasp_IOCTL_CTRL_MUTE_ON16

 5.3.4.7 Mcasp_IOCTL_CTRL_MUTE_OFF.....16

 5.3.4.8 Mcasp_IOCTL_PAUSE.....17

 5.3.4.9 Mcasp_IOCTL_RESUME17

 5.3.4.10 Mcasp_IOCTL_SET_DIT_MODE17

 5.3.4.11 Mcasp_IOCTL_CHAN_TIMEDOUT17

 5.3.4.12 Mcasp_IOCTL_CHAN_RESET18

 5.3.4.13 Mcasp_IOCTL_CNTRL_SET_FORMAT_CHAN.....18

 5.3.4.14 Mcasp_IOCTL_CNTRL_GET_FORMAT_CHAN18

 5.3.4.15 Mcasp_IOCTL_CNTRL_SET_GBL_REGS19

 5.3.4.16 Mcasp_IOCTL_SET_DLB_MODE.....19

 5.3.4.17 Mcasp_IOCTL_ABORT.....19

5.3.5 Channel Deletion.....20

5.3.6 Driver Instance Unbinding/Deletion21

5.4 DATA STRUCTURES21

5.4.1 Internal Data Structures21

 5.4.1.1 Driver Instance Object21

 5.4.1.2 Channel Object22

5.4.2 External Data Structures25

 5.4.2.1 Mcasp_ChanParams.....25

 5.4.2.2 The Mcasp_PktAddrPayload structure.....26

 5.4.2.3 The Mcasp_Params structure26

5.5 SUPPORTED DATA FORMATS27

6 INTEGRATION.....27

6.1 PRE-BUILT APPROACH27

6.2 REBUILD LIBRARY28

7 TEST APPLICATION.....28

LIST OF FIGURES

Figure 1: MCASP Hardware Block Diagram	4
Figure 2: MCASP LLD Software Overview	5
Figure 3: MCASP LLD Driver Architecture.....	6
Figure 4: Device Initialization Sequence	8
Figure 5: Driver Instance Binding.....	11
Figure 6: Create Channel Flow Diagram	13
Figure 7: Control Command Flow	20

1 Scope

This document describes the design of Multichannel Buffered Serial Port Low Level Driver (MCASP LLD). Also, the data types, data structures and application programming interfaces (APIs) provided by the MCASP driver are explained in this document.

2 References

The following references are related to the feature described in this document and shall be consulted as necessary.

No	Referenced Document	Control Number	Description
1	MCASP User Guide	SPRUHH0	KeyStone Architecture MCASP User Guide
2	MCASP LLD Documentation		The MCASP LLD APIs are generated by DOXYGEN and is located in the MCASP package under the “docs” directory in CHM format.
3	EDMA User Guide	SPRUGS5A	Enhanced Direct Memory Access (EDMA3) Controller User Guide

Table 1. Referenced Materials

3 Definitions

Acronym	Description
API	Application Programming Interface
CSL	Chip Support Library
CPU	Central Processing Unit
DMA	Direct Memory Access
DSP	Digital Signal Processor
EDMA	Enhanced Direct Memory Access Controller
FIFO	First In First Out
IP	Intellectual Property
ISR	Interrupt Service Routine
LLD	Low Level Driver
MCASP	Multichannel Buffered Serial Port

Acronym	Description
MMR	Memory Mapped Register
OSAL	Operating System Abstraction Layer
PARAM	Parameter RAM
SOC	System On Chip
SRGR	Sample Rate Generator

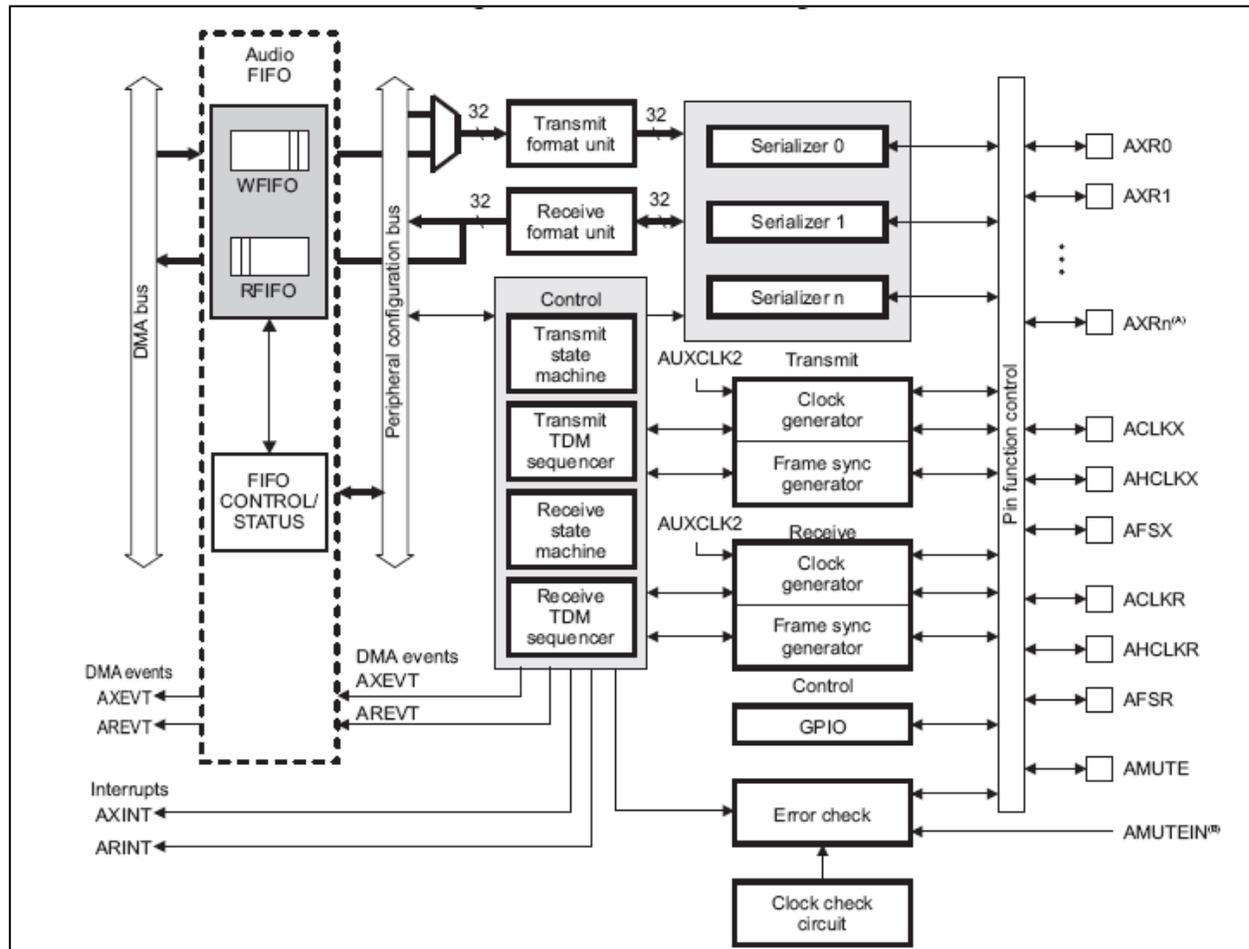
Table 2. Definitions

4 Overview

The multichannel buffered serial port (MCASP) peripheral allows direct interface to other TI DSPs, codecs, and other devices in a system. The primary use for the MCASP is for audio interface purposes. The following sub sections explain the hardware (MCASP peripheral) and software context of the MCASP LLD.

4.1 Hardware Overview

Figure 1: MCASP Hardware Block Diagram



1. The Mcasp is a general purpose serial port optimized for the need of the multichannel audio applications. It supports the TDM and DIT mode of data transfers.
2. The McASP consists of transmit and receive sections that may operate synchronized, or completely independently with separate master clocks, bit clocks, and frame syncs, and using different transmit modes with different bit-stream formats.
3. Extensive error checking and recovery
 - a. Transmit under runs and receiver overruns due to the system not meeting real-time requirements
 - b. Early or late frame sync in TDM mode
 - c. Out-of-range high-frequency master clock for both transmit and receive

- d. DMA error due to incorrect programming.
4. Has hardware FIFO for additional buffering.
5. Has provision for generating the clocks internally or to be sourced from an external source.

4.2 Software Overview

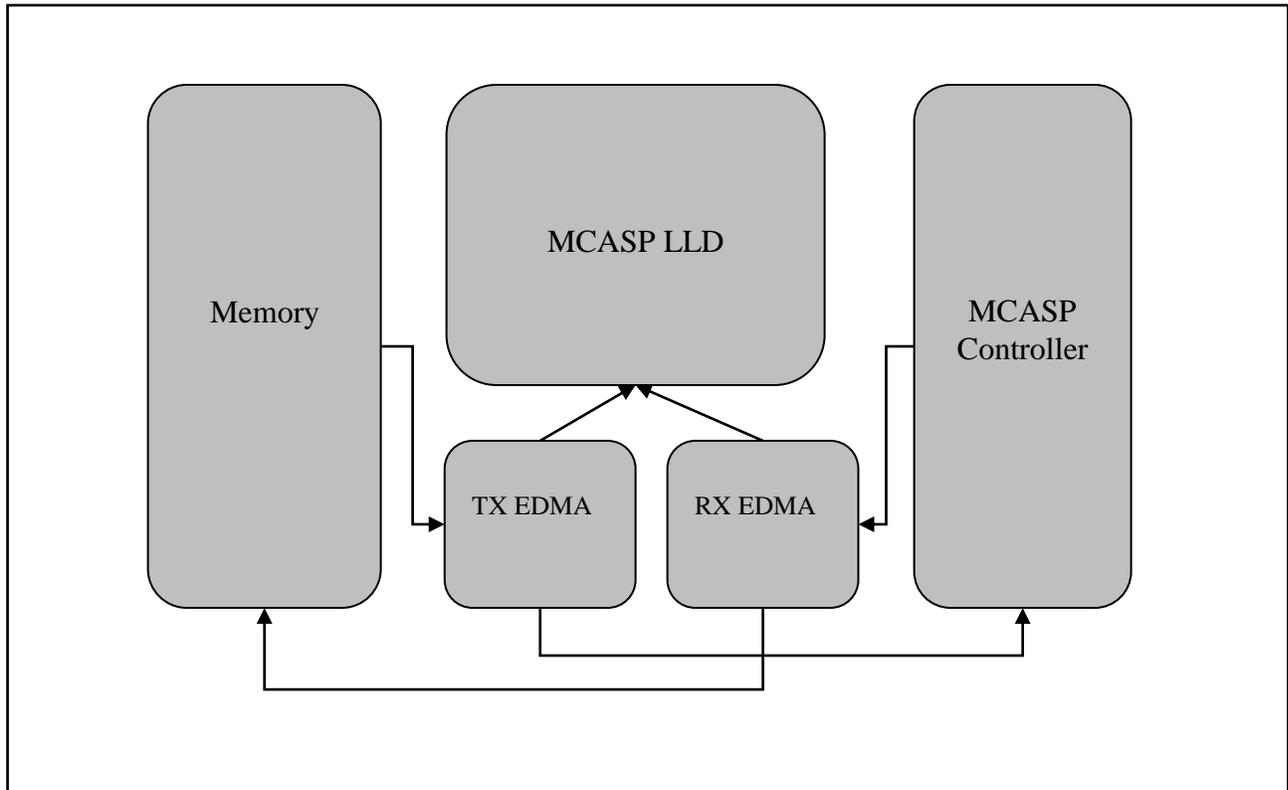


Figure 2: MCASP LLD Software Overview

Figure 2: MCASP LLD Software Overview depicts the various components involved in the transfer of data when the MCASP driver runs on the DSP. Serial data is stored in the memory by DSP e.g. after decoding the audio data. The main function of MCASP driver is to program the EDMA channels to move the data from memory to the MCASP interface on every transfer event from the MCASP (TX path). Similarly, the driver can configure EDMA channels to move data received on MCASP interface to the memory for DSP use (RX path).

The EDMA3 channel controller services MCASP peripheral in the background of DSP operation, without requiring any DSP intervention. Through proper initialization of the EDMA3 channels, they can be configured to continuously service the peripheral throughout the device operation. Each event available to the EDMA3 has its own dedicated channel, and all channels operate simultaneously. The only requirements are to use the proper channel for a particular transfer and to enable the channel event in the event enable register (EER). When programming an EDMA3 channel to service MCASP peripheral, it is necessary to know how data is to be

presented to the DSP. Data is always provided with some kind of synchronization event as either one element per event (non-bursting) or multiple elements per event (bursting).

4.3 Key Features

Following are the key features of MCASP LLD software:

- Multi-instance support and re-entrant driver
- Each instance can operate as a receiver and or transmitter
- Supports multiple data formats
- Can be configured to operate in multi-slot TDM, DSP (used in audio data transfer)
- Mechanisms to transmit desired data (such as NULL tone) when idle

5 Design

This section explains the overall architecture of MCASP device driver, including the device driver functional partitioning as well as run-time considerations. The MCASP LLD driver provides well-defined API layers which allow applications to use the MCASP peripheral to send and receive data.

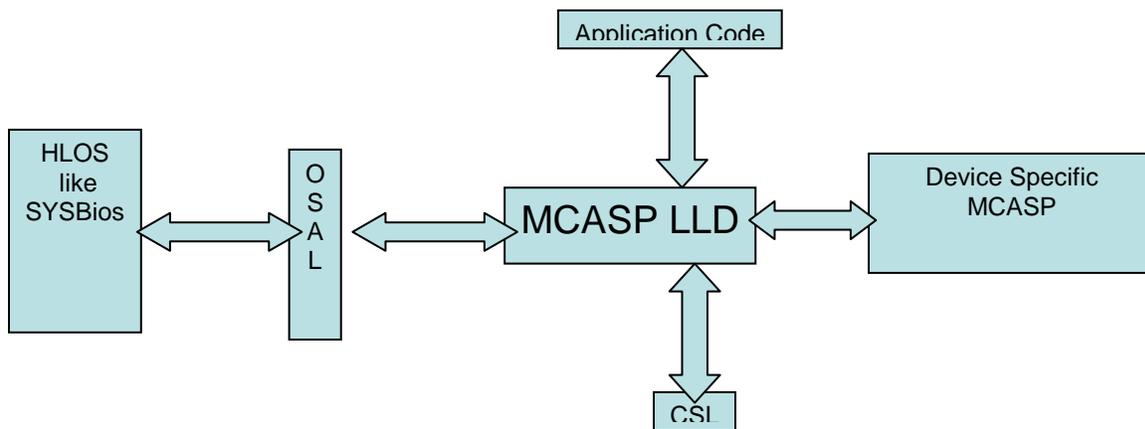


Figure 3: MCASP LLD Driver Architecture

The [Figure 3: MCASP LLD Driver Architecture](#) illustrates the following key components:

1.) MCASP Device Driver

This is the core MCASP device driver. The device driver exposes a set of well-defined APIs which are used by the application layer to send and receive data via the MCASP peripheral. The driver also exposes a set of well-defined OS abstraction APIs which are used to ensure that the driver is OS independent and portable. The MCASP driver uses the CSL MCASP register layer for all MCASP MMR access. The MCASP driver also interfaces with the EDMA3 library to be able to transfer data to and from MCASP peripheral and data memory.

2.) Device Specific MCASP Layer

This layer implements a well defined interface which allows the core MCASP driver to be ported on any device which has the same MCASP IP block. This layer may change for every device.

3.) Application Code

This is the user of the driver and its interface with the driver is through the well-defined APIs set. Application users use the driver APIs to send and receive data via the MCASP peripheral.

4.) Operating System Abstraction Layer (OSAL)

The MCASP LLD is OS independent and exposes all the operating system callouts via this OSAL layer.

5.) CSL Register Layer

The CSL register layer is the IP block memory mapped registers which are generated by the IP owner. The MCASP LLD driver directly accesses the MMR registers.

5.1 MCASP Driver Initialization

The MCASP Driver initialization API needs to be called only once and it initializes the internal driver data structures like device objects. Application developers need to ensure that they call the MCASP Driver Init API before they call the MCASP Device Initialization.

The following API is used to initialize the MCASP Driver.

```
int32_t mcasplnit (void)
```

The function returns **MCASP_STATUS_COMPLETED** on success indicating that the MCASP driver internal data structures have been initialized correctly.

5.2 MCASP Peripheral Configuration

The MCASP driver provides a sample implementation sequence which initializes the MCASP IP block. The MCASP Device initialization API is implemented as a sample prototype:

```
void Mcaspldevice_init (void)
```

The function initializes all the instance specific information like base address of instance CFG registers, FIFO address for the instance, TX and RX CPU event numbers, TX and RX EDMA event numbers etc. The function also sets the inUse field of MCASP instance module object to FALSE so that the instance can be used by an application which will create it. Mute buffers are also initialized. Please refer to the figure below for the typical control flow during the device initialization.

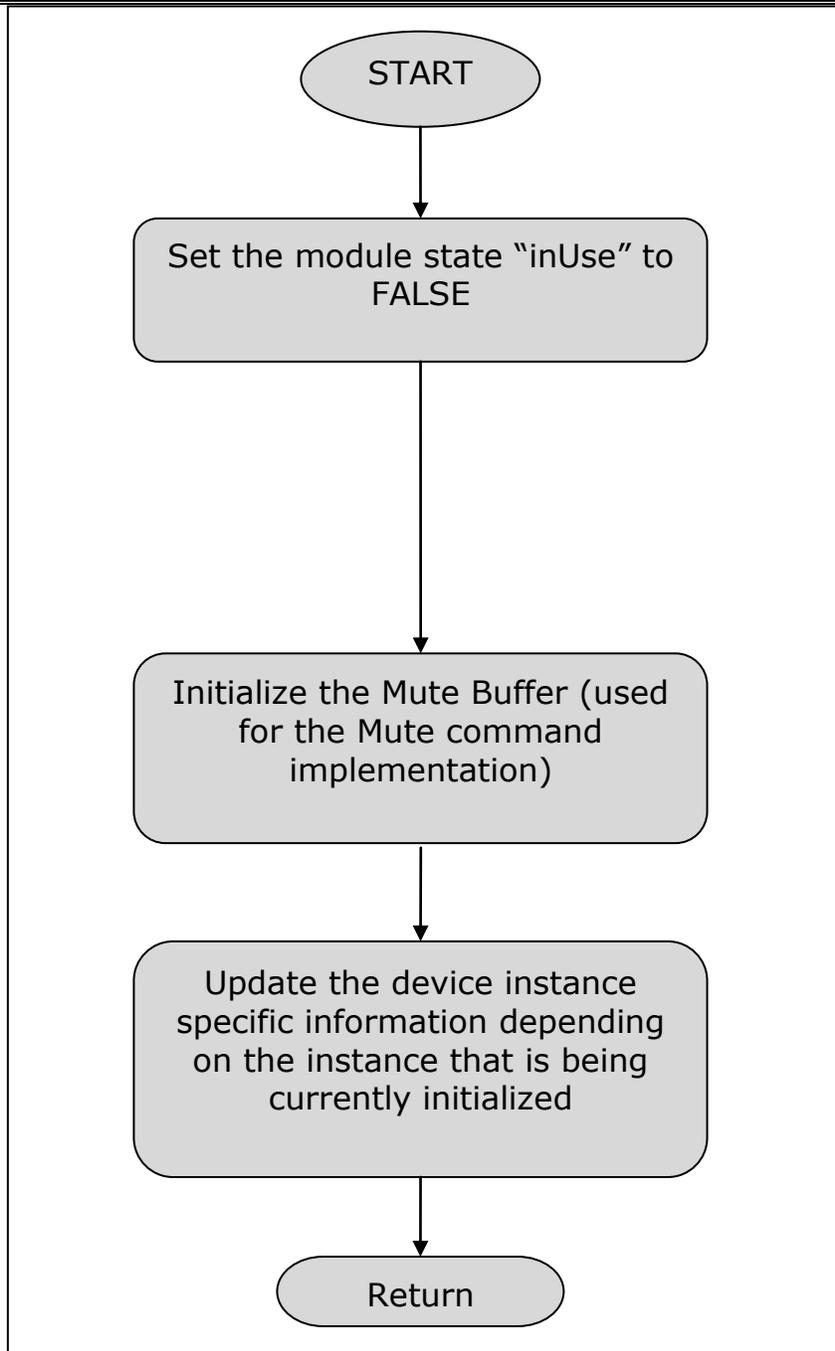


Figure 4: Device Initialization Sequence

The Figure 4: Device Initialization Sequence depicts the typical control flow during the initialization of the MCASP device.

This implementation is **sample only** and application developers are recommended to modify it as deemed necessary. The initialization sequence is **not** a part of the MCASP driver library. This was done because the MCASP Device Initialization sequence has to be modified and customized by application developers. If the initialization sequence was a part of the MCASP driver then it

would require the driver to be rebuilt. Moving this API outside the driver realm solves this issue. The MCASP Device Initialization API should only be called after calling the MCASP Device Init API. Failure to do so will result in unpredictable behaviors.

5.3 MCASP Driver External Interface (Public APIs)

The following table outlines the basic interfaces provided by MCASP LLD.

Function	Description
mcaspBindDev	The mcaspBindDev function is called by the application after MCASP device initialization. The mdBindDev performs following actions: <ul style="list-style-type: none"> ❖ Acquire the device handle for the specified instance of MCASP on the SOC. ❖ Configure the MCASP device instance with the specified parameters (or default parameters, if there is no external configuration).
mcaspUnBindDev	The mcaspUnBindDev function is called to delete an instance of the Mcas driver. It will unroll all the changes done during the bind operation and free all the resources allocated to the MCASP.
mcaspCreateChan	The mcaspCreateChan function creates a TX or RX channel on the specified MCASP instance. Application has to specify the mode in which the channel has to be created through the “ mode ” parameter. The MCASP driver supports only two modes of channel creation (input and output mode) for every device instance. It performs following actions: <ul style="list-style-type: none"> ❖ The required EDMA channel and spare PARAM sets are acquired and configured. ❖ The required TX or RX sections (clocks, SRGR, frame sync etc.) are setup.
mcaspDeleteChan	The mcaspDeleteChan deletes a channel created on a MCASP instance. It frees all the resources allocated during the creation of the channel.
mcaspSubmitChan	The mcaspSubmitChan is invoked with the appropriate channel handle and IOBuf (aka frame) containing the operation to be performed and required parameters needed for programming the EDMA channels.
mcaspAppCallback	This function is the callback function routine called when any RX or TX transfer is completed.
mcaspControlChan	The mcaspControlChan function is used to issue a control command to the MCASP driver. Please refer to the list of control commands supported by the MCASP driver. <ul style="list-style-type: none"> ❖ Typical commands supported are PAUSE, RESUME, STOP, START etc.

5.3.1 Driver Instance Binding

The binding function (**mcaspBindDev**) of the MCASP driver is called to allocate and configure a MCASP instance as specified by **devId**. Each driver instance corresponds to one hardware instance of the MCASP. The function performs following actions:

- Check if the instance being created is already in use by checking “**inUse**”.
- Update the instance object with the user supplied parameters.
- Initialize all the channel objects (TX and RX) with default parameters.
- Initialize queues to hold the pending frames and currently executing frames (floating queue).
- Configure the MCASP to receive the Frame Sync and bit clocks either externally or internally for both receiver and transmitter depending on the user supplied parameters.
- Return the device handle.

The driver binding operation expects the following parameters:

1. Pointer to hold the function returned device handle.
2. Instance number of the MCASP instance being created.
3. Pointer to the user provided device parameter structure required for the creation of device instance. The user provided device parameter structure will be of type “**Mcasp_Params**”.

Please refer the [Figure 5: Driver Instance Binding](#) below for the control flow of driver Bind operation.

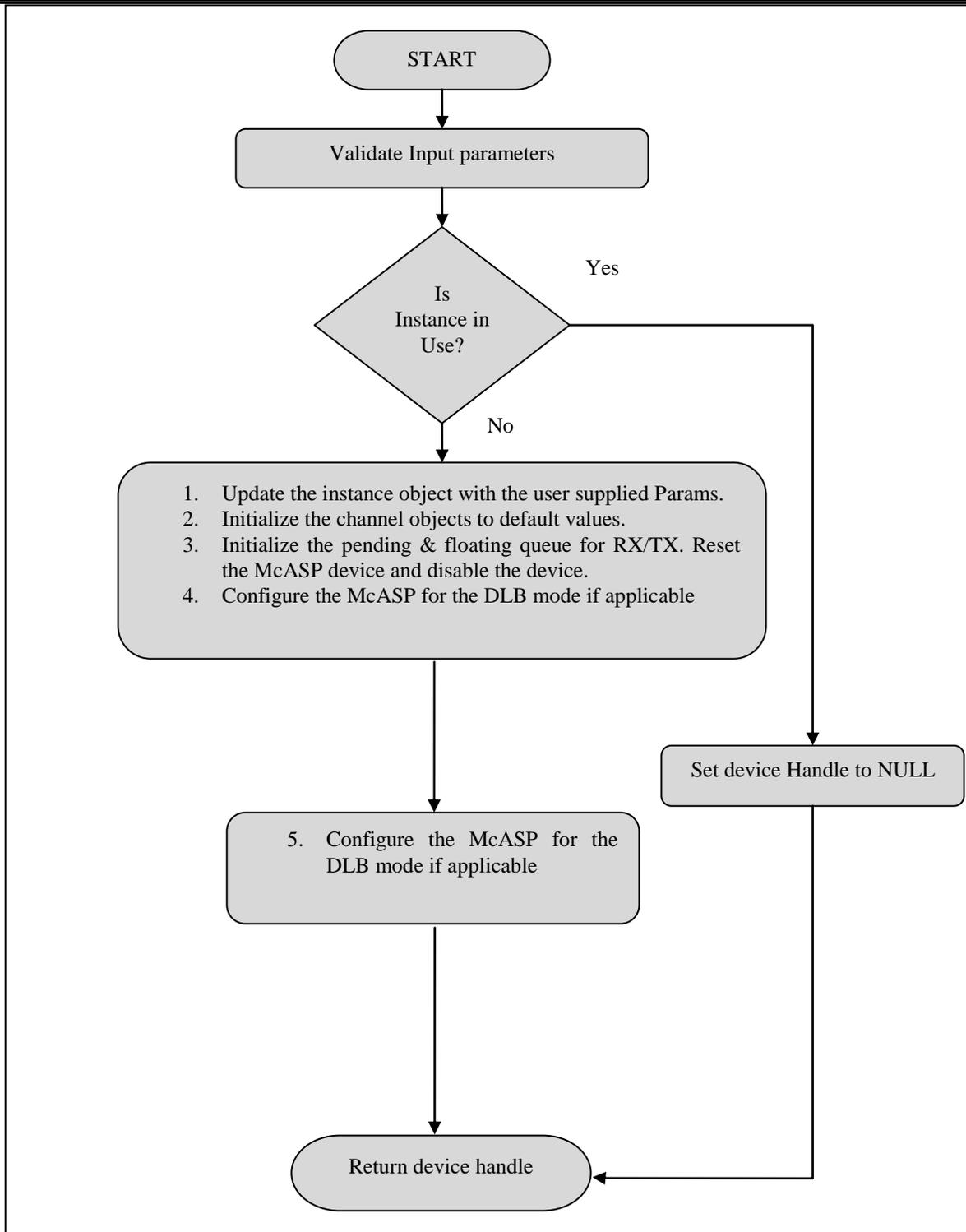


Figure 5: Driver Instance Binding

5.3.2 Channel Creation

Once the application has created a device instance, it needs to create a communication channel for transactions with the underlying hardware. As such a channel is a logical communication interface between the driver and the application. The driver allows at most two channels per MCASP instance to be created which are a transmit channel (TX path e.g. audio playback or data transmission) and a receive channel (RX path e.g. audio recording or data reception). The application can create a communication channel by calling **mcaspCreateChan** function. The application should call **mcaspCreateChan** with the appropriate “**mode**” (**MCASP_MODE_OUTPUT** or **MCASP_MODE_INPUT**) parameter for the type of the channel to be created.

The application needs to supply the parameters which will characterize the features of the channel e.g. number of slots, slot width etc. The application can use the “**Mcasp_ChanParams**” structure to specify the parameters to configure the channel.

The **mcaspCreateChan** function performs the following actions:

- Validates the input parameters given by the application.
- Checks if the requested channel is already opened or not. If it is already opened then the driver will flag an error to the application else the requested channel will be allocated.
- Updates the appropriate channel object with the user supplied parameters.
- MCASP is configured with the appropriate word width.
- EDMA parameters for the requested channel are setup.
- If the global error callback function registration is enabled, the appropriate user supplied function is registered to be called in case of an error.
- If the channel creation fails then it will perform a cleanup and also free all the resource allocated by it till now.
- If the complete process of channel creation is successful, then it will return a unique channel handle to the application. This handle should be used by the application for further transactions with the channel. This handle will be used by the driver to identify the channel on which the transactions are being requested.

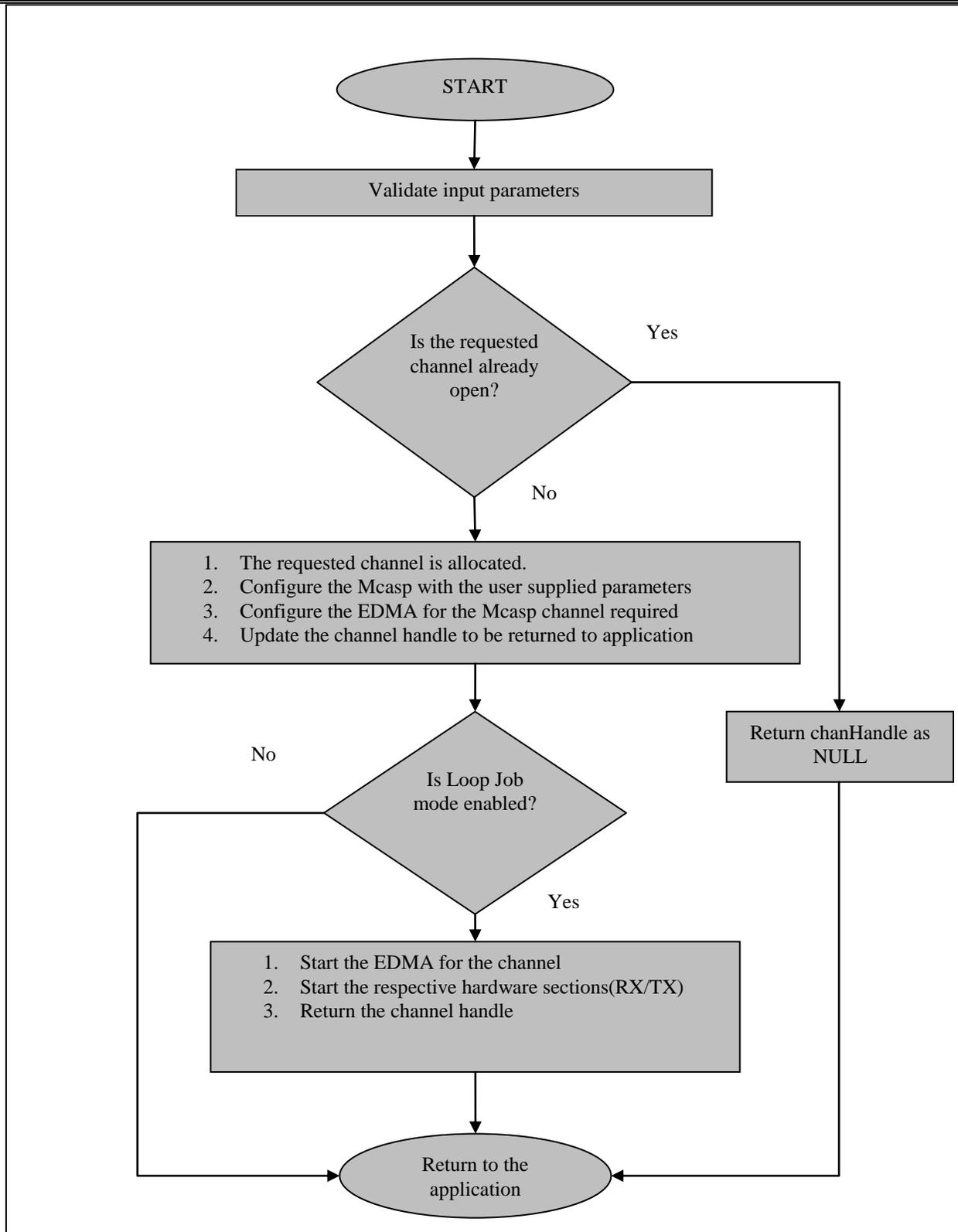


Figure 6: Create Channel Flow Diagram

5.3.3 I/O Frame Processing

MCASP driver provides **mcaspSubmitChan** interface to submit **ioBufs** (frames) for the I/O transactions to be performed. Application invokes this API for data transfer using MCASP. This API submits a **Mcasp_IOBuf** frame containing all the transfer parameters needed by the driver to program the underlying hardware for data transfer. The **mcaspSubmitChan** function handles the command code passed to it as part of the **Mcasp_IOBuf** structure.

The **mcaspSubmitChan** function performs the following actions:

- The input **Mcasp_IOBuf** frame is validated.
- If the driver has sufficient frames then the current frame is loaded in to the pending queue.
- Otherwise the frame is programmed into the link PARAMs of the EDMA.
- In NON LOOP JOB mode, the first frame is always loaded in to the main transfer channel. The subsequent two frames are loaded into the spare PARAM sets of the EDMA. Also if this is the first frame for the driver then the clocks are started as per the configuration of the channel. Any other frames after this are loaded into the pending queue. These frames will be loaded by the EDMA callback into the appropriate PARAM set of the EDMA.

5.3.3.1 Asynchronous I/O Mechanism

The MCASP driver supports asynchronous I/O mechanism. In this mechanism, multiple I/O requests can be submitted by the application without causing it to block while waiting for the previous I/O requests to complete. Application can submit multiple I/O requests using **mcaspSubmitChan** API. The application callback function registered during the transfer request submission will be called upon transfer completion by the driver. The driver internally will queue the I/O frames submitted to support the asynchronous I/O functionality.

5.3.4 Control Commands

MCASP driver implements device specific control functionality which may be useful for any application, which uses the MCASP driver. Application may invoke the control functionality through a call to **mcaspControlChan**. MCASP driver supports the following control functionality. The following sections list the IOCTL commands supported by the McASP driver.

5.3.4.1 Mcasp_IOCTL_DEVICE_RESET

Command	Mcasp_IOCTL_DEVICE_RESET
Parameters	None

The application issues this command to reset the Mcasp device. On receiving this command the Mcasp aborts all the current pending IO requests and resets the Mcasp.

Note: This command resets the entire Mcasp instance irrespective of the channel to which the command is issued.

5.3.4.2 Mcasp_IOCTL_CNTRL_AMUTE

Command	Mcasp_IOCTL_CNTRL_AMUTE
Parameters	Value to write to AMUTE register

The application issues this command to control the AMUTE pin of the McASP device. The application also needs to specify the value to be written to the AMUTE register.

5.3.4.3 Mcasp_IOCTL_START_PORT

Command	Mcasp_IOCTL_START_PORT
Parameters	None

This IOCTL function allows the application to start the state machine of the required channel. The McASP driver has two channels receive and transmit channel. This command starts the state machine of the channel for which the start command has been issued.

5.3.4.4 Mcasp_IOCTL_STOP_PORT

Command	Mcasp_IOCTL_STOP_PORT
Parameters	None

This IOCTL function allows the application to stop the state machine of the required channel. The McASP driver has two channels, receive and transmit channels. This command stops the state machine of the channel for which the command has been issued.

5.3.4.5 Mcasp_IOCTL_QUERY_MUTE

Command	Mcasp_IOCTL_QUERY_MUTE
Parameters	Pointer to the variable to hold the AMUTE register value

This command is used by the application to query the value of the McASP AMUTE register. The application provides a pointer where the queried value is updated

5.3.4.6 Mcasp_IOCTL_CTRL_MUTE_ON

Command	Mcasp_IOCTL_CTRL_MUTE_ON
Parameters	None.

This command mutes the Mcasp device i.e. only zeros will be sent instead of an audio stream.

5.3.4.7 Mcasp_IOCTL_CTRL_MUTE_OFF

Command	Mcasp_IOCTL_CTRL_MUTE_OFF
Parameters	None

This command is used to “un-mute” the previously muted Mcasp device. In case of trying to un-mute a channel which is not muted, an error is given by the IOM Driver.

5.3.4.8 Mcasp_IOCTL_PAUSE

Command	Mcasp_IOCTL_PAUSE
Parameters	None

This command sets the McASP device in to pause i.e. no more IO packets are processed by the McASP. All the new requests will be queued up in the IOM Driver.

5.3.4.9 Mcasp_IOCTL_RESUME

Command	Mcasp_IOCTL_RESUME
Parameters	None

This command is used to resume the McASP device which is paused previously. In the case that the Mcasp is not in a paused state the IOM Driver raises an error.

5.3.4.10 Mcasp_IOCTL_SET_DIT_MODE

Command	Mcasp_IOCTL_SET_DIT_MODE
Parameters	Value to write to DITCTL register

This command is used to modify the Mcasp audio data transport protocol to the DIT mode.

5.3.4.11 Mcasp_IOCTL_CHAN_TIMEDOUT

Command	Mcasp_IOCTL_CHAN_TIMEDOUT
---------	---------------------------

Parameters	None
------------	------

This command is to be called in case a timeout is encountered during a channel operation. This command aborts the timed out channel.

5.3.4.12 Mcasp_IOCTL_CHAN_RESET

Command	Mcasp_IOCTL_CHAN_RESET
Parameters	None

This command is used by the application to reset a McASP channel

5.3.4.13 Mcasp_IOCTL_CNTRL_SET_FORMAT_CHAN

Command	Mcasp_IOCTL_CNTRL_SET_FORMAT_CHAN
Parameters	Pointer to the new “Mcasp_HwSetupData” structure

This command is used to modify the channel settings of the McASP channel. It configures the Mcasp channel with the new hardware set up data sent by the application.

5.3.4.14 Mcasp_IOCTL_CNTRL_GET_FORMAT_CHAN

Command	Mcasp_IOCTL_CNTRL_GET_FORMAT_CHAN
Parameters	Pointer to the “Mcasp_HwSetupData” structure to hold the data

The application can use this command to get the information about the current channel. The application needs to provide the pointer to the “Mcasp_HwSetupData” structure to hold the data.

5.3.4.15 Mcasp_IOCTL_CNTRL_SET_GBL_REGS

Command	Mcasp_IOCTL_CNTRL_SET_GBL_REGS
Parameters	Pointer to the “Mcasp_HwSetup” structure

The application can use this command to set the global control register. The application needs to send the pointer to the new “Mcasp_HwSetup” data structure that needs to be programmed.

5.3.4.16 Mcasp_IOCTL_SET_DLB_MODE

Command	Mcasp_IOCTL_SET_DLB_MODE
Parameters	DLB mode enable or disable

This command is used to set the McASP in to the loopback mode.

5.3.4.17 Mcasp_IOCTL_ABORT

Command	Mcasp_IOCTL_ABORT
Parameters	None

The application issues this command to abort the pending requests of the channel. This IOCTL aborts all the pending request of the channel and stops the state machine. The EDMA transfer is also stopped.

The typical control flow for the MCASP control function is as given below.

- Validate the command sent by the application.
- Check if the appropriate arguments are provided by the application for the execution of the command.
- Process the command and return the status back to the application.

The basic control flow for the handling of the control commands for the driver is shown in [Figure 7: Control Command Flow](#). Please note that the individual command handling is not detailed here.

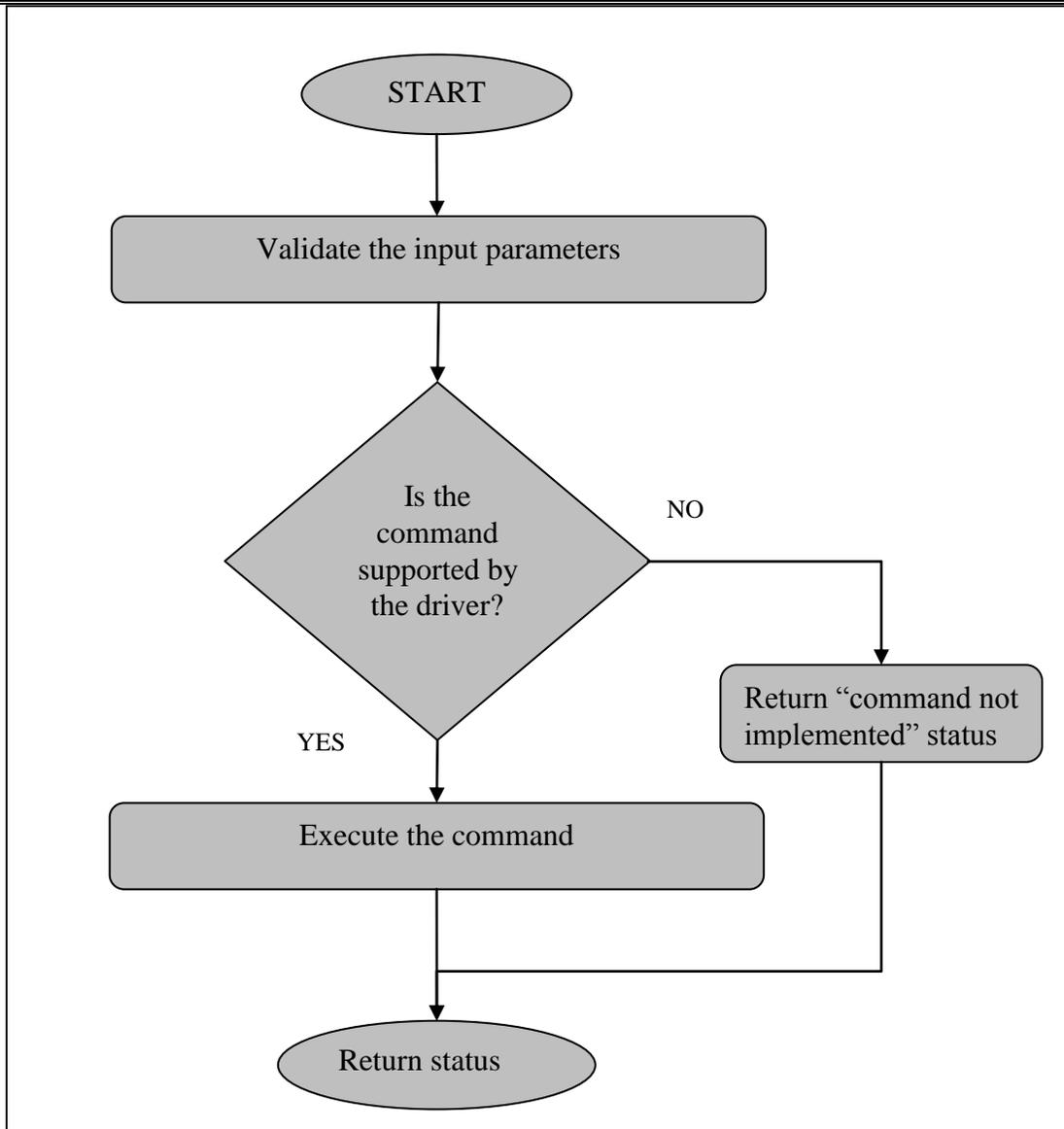


Figure 7: Control Command Flow

5.3.5 Channel Deletion

Once a channel has completed all the transactions it can be closed so that all the resources allocated to the channel can be freed. The driver provides `mcaspDeleteChan` API to delete a previously created MCASP channel for an instance. The actions performed during the channel deletion are as follows:

- The channel to be deleted is reset.
- The reset operation aborts all the packets in the pending queue and also the packets in the current active queue.
- The EDMA transfer for this channel is disabled.
- The MCASP state machines are stopped.
- The interrupt handlers are unregistered.

- All the spare PARAM sets of the EDMA are freed.
- The status of the channel is updated to DELETED.

5.3.6 Driver Instance Unbinding/Deletion

The MCASP driver provides **mcaspUnBindDev** interface to delete a driver instance. The function de-allocates all the resources allocated to the instance object during the driver binding operation. The operations performed by the unbind operation are as listed below:

- Check if both the TX and the RX channels are closed.
- Update the instance object.
- Set the status of the driver instance to “DELETED”.
- Set the status of the instance “inUse” to FALSE (so that instance can be used again).

5.4 Data Structures

5.4.1 Internal Data Structures

5.4.1.1 Driver Instance Object

This structure is the MCASP driver’s internal data structure. This data structure is used by the driver to hold the information specific to the MCASP instance. There will be one unique instance object for every instance of the MCASP controller supported by the driver.

S.No	Structure Elements (<i>McasP_Object</i>)	Description
1	<i>instNum</i>	Current instance number of the Mcasp
2	<i>devState</i>	Preserve the current state of the driver (Create/Deleted etc)
3	<i>isDataBufferPayloadStructure</i>	Whether the application buffer to be interpreted as payload structure.
4	<i>mcaspHwSetup</i>	McasP device hardware information for initializing
5	<i>hwiNumber</i>	Interrupt number used by the Mcasp
6	<i>enablecache</i>	Flag to enable/disable the usage of the cache
7	<i>stopSmFsXmt</i>	Flag to stop the transmit state machine
8	<i>stopSmFsRcv</i>	Flag to stop the receive state machine

9	<i>XmtObj</i>	Transmit channel object
10	<i>RcvObj</i>	Receive Channel object
11	<i>HwlInfo</i>	Structure holding the instance specific information like base address etc
12	<i>serStatus</i>	Status of each serializers (e.g. Free/transmit/receive)
13	<i>isrSwiObject</i>	SWI to handle the interrupts
14	<i>fifoSwiObject</i>	SWI object for the swi used to handle the FIFO
15	<i>retryCount</i>	value to be used when waiting for the TX empty
16	<i>loopJobMode</i>	loop job mode is always disabled
17	<i>pscPwrMEnable</i>	PSC control enable/disable

Comments

1. The MCASP Driver works only in the EDMA mode of operation.
2. One instance object represents one instance of the driver.

Constraints

None

See Also

Mcasp_ChannelObj

5.4.1.2 Channel Object

This structure is the MCASP driver’s internal data structure. This data structure is used by the driver to hold the information specific to a channel. There will be at most two channels supported per MCASP instance (one for TX and one for RX). It is used to maintain the information pertaining to the channel like the current channel state, callback function etc. This structure is initialized by `mcaspCreateChan` and a pointer to this is passed down to all other channel related functions. Lifetime of the data structure is from its creation by `mcaspCreateChan` till it is invalidated (deleted) by `mcaspDeleteChan`.

S.No	Structure Elements (<i>Mcasp_ChannelObj</i>)	Description
1	<i>chanState</i>	Preserve the current state of the driver (Open/closed etc)

2	<i>mode</i>	Mode of operation of the channel
3	<i>devHandle</i>	Pointer to the Mcasp_Object structure
4	<i>cbFxn</i>	Callback function to be called on completion of an IO operation
5	<i>cbArg</i>	Argument to be passed to the callback function
6	<i>queueReqList</i>	List to handle all the pending IO packets
7	<i>queueFloatingList</i>	List to handle the currently processed IO requests
8	<i>noOfSerAllocated</i>	Number of serializers allocated for this channel
9	<i>channelOpMode</i>	Audio data transport protocol (DIT/TDM)
10	<i>isDmaDriven</i>	Option to check if the channel is DMA driven
11	<i>dataQueuedOnReset</i>	Data queued in the channel.
12	<i>intStatus</i>	Interrupt status
13	<i>dataPacket</i>	Current IO packet pointer
14	<i>tempPacket</i>	Temporary IO packet pointer
15	<i>isTempPacketValid</i>	Flag to indicate whether the tempPacket field holds an valid packet.
16	<i>userDataBufferSize</i>	Size of the application given buffer
17	<i>submitCount</i>	Number of IO requests pending in the channel
18	<i>indexOfSersRequested[]</i>	Index of the serializers requested by the channel.
19	<i>edmaHandle</i>	Handle to the EDMA driver
20	<i>xferChan</i>	EDMA transfer channel
21	<i>tcc</i>	EDMA Transfer channel
22	<i>pramTbl[2]</i>	Spare channels of EDMA used for linking
23	<i>pramTblAddr[2]</i>	Physical address of EDMA

		channels used for linking
24	<i>nextLinkParamSetToBeUpdated</i>	Element holding the next paramset to be linked
25	<i>loopjobUpdatedinParamset</i>	Flag to check if the loop job is updated in the param set (Not used as loopjob mode is permanently disabled)
26	<i>cpuEventNum</i>	Cpu interrupt number
25	<i>xferinProgressIntmode</i>	Flag to indicate that transfer is in progress in the interrupt mode
26	<i>loopJobBuffer</i>	Pointer to Loop job buffer to be used when the Mcasp is idle (Not used as loopjob mode is permanently disabled)
27	<i>loopJobLength</i>	Length of the loop job to be used for each serializer (Not used as loopjob mode is permanently disabled)
28	<i>roundedWordWidth</i>	Length of the word to be transferred in EDMA
29	<i>currentDataSize</i>	Current transfer size
30	<i>bMuteON</i>	Flag to indicate if the mute is on for this channel
31	<i>paused</i>	Flag to indicate if the channel is paused
32	<i>edmaCallback</i>	Pointer to the edma callback function
33	<i>gblErrCbK</i>	Pointer to the call back to be called in case of error
34	<i>nextFlag</i>	Flag used to check if the channel state machine can be stopped
35	<i>currentPacketErrorStatus</i>	Current packet Error status is maintained here
36	<i>enableHwFifo</i>	whether the FIFO has to be enabled for this channel
37	<i>isDataPacked</i>	flag to indicate if the buffer data needs to be packed
38	<i>userLoopJobLength</i>	Length of the user supplied loop job buffer (Not used as

		loopjob mode is permanently disabled)
39	<i>dataFormat</i>	Application supplied buffer format
40	<i>userIntValue</i>	User supplied mask for the interrupts to be enabled
41	<i>userLoopJob</i>	Option to indicate if the user loop job is used or driver loop job (Not used as loopjob mode is permanently disabled)

Comments

Constraints

None

See Also

Mcasp_Object

5.4.2 External Data Structures

5.4.2.1 Mcasp_ChanParams

This structure is used to supply user parameters during the creation of the channel instance. During the creation of the channel, user needs to supply the above structure with the appropriate parameters as per the required mode of operation. The structure is defined as below:

S.No	Structure Elements (<i>Mcasp_ChanParams</i>)	Description
1	<i>noOfSerRequested</i>	Number of serializers requested
2	<i>indexOfSersRequested[]</i>	Index of the requested serializers
3	<i>mcaspSetup</i>	Pointer to the Mcasp hardware set up structure
4	<i>isDmaDriven</i>	Flag to indicate if the channel is DMA driven
5	<i>channelMode</i>	Audio transport protocol to be used(DIT/DTM)
6	<i>wordWidth</i>	Size of the data to transferred
7	<i>userLoopJobBuffer</i>	Loop job buffer to be used

		(Not used as loopjob mode is permanently disabled)
8	<i>userLoopJobLength</i>	Loop job buffer length (Not used as loopjob mode is permanently disabled)
9	<i>edmaHandle</i>	Handle to the EDMA driver
10	<i>gblCbk</i>	Pointer to the callback function to be called in case of Errors
11	<i>noOfChannels</i>	Number of channels to be transmitted (used only in case of TDM mode).
12	<i>dataFormat</i>	Format of the application supplied buffer
13	<i>enableHwFifo</i>	Option to enable the Hardware FIFO
14	<i>isDataPacked</i>	flag to indicate if the buffer data needs to be packed

See Also

Mcasp_DataConfig

5.4.2.2 The *Mcasp_PktAddrPayload* structure

This is the format of the audio data to be sent by the application in case of Using DIT mode.

S.No	Structure Elements (<i>Mcasp_PktAddrPayload</i>)	Description
1	<i>chStat</i>	Channel status ram info
2	<i>userData</i>	User information
3	<i>writeDitParams</i>	Whether the DIT params are to be written
4	<i>addr</i>	Actual address of the buffer to be transferred

5.4.2.3 The *Mcasp_Params* structure

S.No	Structure Elements (<i>Mcasp_Params</i>)	Description
------	---	-------------

1	<i>enablecache</i>	Whether cache has to be used.
2	<i>hwiNumber</i>	Hwi number to be used by the Mcasp device
3	<i>isDataBufferPayloadStructure</i>	Whether the buffer is to be interpreted as payload structure or normal buffer
4	<i>mcaspHwSetup</i>	The Mcasp hardware initialization structure
5	<i>pscPwmEnable</i>	Option to enable or disable the PSC control

5.5 Supported Data Formats

6 Integration

The MCASP LLD depends on the following components:

- a. CSL
- b. EDMA3 LLD

These components need to be installed before the MCASP driver can be integrated. The MCASP driver is released in source code and in pre-built library. Applications can decide how to use the MCASP driver.

The MCASP Driver release notes indicate the version of the above components which that release is dependent upon. The next steps use the version numbers for illustrative purpose only.

6.1 Pre-built approach

In this approach, the application developers can decide to use the MCASP driver pre-built libraries as is. The following steps need to be done:

- a. The application developers modify their application configuration file to use the MCASP package.

```
var Mcasp = xdc.loadPackage('ti.drv.mcasplld');
```

- b. Ensure that the XDCPATH is configured to have the path to the PDK package
- c. This implies that XDC Configuration scripts will link the application using the MCASP Driver libraries (**Module.xs**)
- d. The application authors need to provide an OSAL implementation file for MCASP and ensure that this is linked with the application; failure to do so will result in linking

errors. Please refer to the MCASP OSAL header file (`mcasp_osal.h`) for more information on the API's which need to be provided.

6.2 Rebuild library

In this approach, the application developers can decide to use the MCASP driver source code and add these files to the application project to rebuild the MCASP driver code base. The following steps need to be redone:

- a. Application developers should port the file “`mcasp_osal.h`” to their operating system environment. *Developers are recommended to create a copy of this file and place it in their application directory.* They should use the file which is provided in the MCASP installation only as a template. The goal here should be to map the `Mcasp_osalXXX` macros to the OS calls directly thus reducing the overhead of an API callout. E.g.

```
#define Mcasp_osalCreateSem()      (Void*)Semaphore_create(0, NULL, NULL)
```

- b. Application developers should port the file “`mcasp_types.h`” to the application environment. *Developers are recommended to create a copy of this file and place it in their application directory.*
- c. Append the include path to the top level MCASP package directory i.e. if the MCASP package is installed in `C:\Program Files\Texas Instruments\mcasp_C6657_1_0_0_0`; then make sure the include path is configured as `C:\Program Files\Texas Instruments\mcasp_C6657_1_0_0_0\packages`
- d. Add the MCASP driver files listed in the `src` directory to the application build files

The approach above is highlighted in the MCASP `example` directory.

7 Test Application

The test application is an audio loopback example. This programs the AIC3104 codec on AM572x GP-EVM to accept analog input and output analog. The code to program the codec is present in the following files under the directory

`mcasp\example\evmAM572x\AIC31_Stereo_Loopback\src\`

```
Aic31.c  
aic31_if.c  
audio_evmInit.c  
audioSample_io.c  
audioSample_main.c  
codec_if.c  
mcasp_osal.c
```

The test example receives the audio samples received on the McASP port through the AIC3104 codec EVM and plays them back through the audio output port of the EVM.